CHANGE REPORT

NP Studios – Team 8

Team Members
Lucy Ivatt, Jordan Spooner, Alasdair Pilmore-Bedford, Matthew Gilmore, Bruno Davies, Cassandra Lillystone

Our Formal Approach to Change Management

Our approach to change management can be seen in the table (https://npstudios. github.io /files/ChangeManagementTable.pdf) and is described in more detail below. As a group we acted as the **change committee;** we decided whether a change is necessary and if it will be implemented or not. As we have inherited another group's game, there may be aspects of the brief or their requirements that they missed out. For example, as mentioned in Dicy Cat's implementation report from assessment 2, they did not manage to implement ET fortresses having unique statistics. This, along with any changes communicated to us by the customer, would initiate a **change request**. The requests would then be reviewed, and a priority determined with high priority tasks being implemented first. For example, the requirement **UR_COLOUR_ACCESSIBILITY** which specifies the game should contain different colour schemes is additional to the brief so was given a lower priority. Once all changes were reviewed, we then added them to a tracking document (https://npstudios.github.io/files/ChangeTracker.pdf) containing the following headings: Change Description, Requirement(s) the change related to, Change Builder, Status and Branch.

The **change builder** is the member of the team who is responsible for the change and implements this into the product. Before assigning changes to members of the team, we grouped similar changes together, for example we assigned 'Make sure fortresses do different damage' and 'After a certain amount of time, damage from fortresses increases' to the same group. Each group is then assigned to someone to implement. We believe grouping the changes helps to prevent conflicts in the code and makes the overall process of implementing the changes run smoother. Every member of the group will be a **change builder** so we can prioritise implementation of the changes in-case it is more complex and time consuming then we originally thought.

The change tracker also contains the titles branch and status. For each new distinct change or addition, we created a new branch so that each team member could work without frequent merge conflicts. This also allowed us to keep the master branch free of untested changes, so we always had a concrete and fully working restore point if needed. The status of the change would be either not started, in progress or complete to help us track our progress and make sure we stayed on our timeline. Once the change was complete, the **change builder** would make a pull request which would have to be reviewed by the team before merging to the master branch.

Deliverables and Documentation

When altering or adding to any documents, we made sure to highlight any changes we made clearly. as well as adding detailed comments where necessary. For any documents that have been changed, we have uploaded a new version showing these changes such as the additional updated requirements table (https://npstudios.github.io/files/NewandChangedRequirements.pdf). We primarily used a colour coding method to clearly show changes and additions and used ~~strikethrough~~ to show deletions in this document, making it obvious what changes we have made. Another example is our additional UML diagram. We made this to show how our new classes relate to the already existing classes, ensuring our code meets the overall architecture.

Code

By continuing to use git and GitHub as version control for our project, we can easily use this to keep track of changes to the game. As each change/addition has its own branch and by using pull requests we can easily compare the changed branch and master and ensure that we have fully documented any new code using our template (**Figure 1, Figure 2**) . Every change we make will be given a unique ID and will be enclosed between a beginning and end comment, to make it less ambiguous as to what code was edited. We also ensured that any new classes and methods we created had comprehensive JavaDocs added too. Once the change was merged with the master branch, we then updated the change tracker to say the change has been completed.

Changes to Testing

We have decided to remove the debug features, originally included by Dicy Cat, in the game because it caused a memory leak (see our implementation report for more detail). We believe this feature can be removed as we can take more effective approaches to testing. We continued to use their method of thoroughly play-testing the game because we believe this is one effective way of testing. It allows us to rule out any glitches or bugs that the player may encounter, as well as being able to identify requirements we have missed from the implementation. This, however, didn't act as our main method of testing.

In addition to this, we have decided to implement JUnit testing. This gives us a self-validating way to isolate the requirements and test the actual outcome based on the expected input and outcome. To accomplish this, we ran the tests 'headless', which is where the game is simulated without showing. This allowed us to test classes that, for example, needed instantiated textures in order to be created. Alongside this, we mocked any dependencies that were found within classes. Providing insurance that the unit test results were solely from the method in question, However, headless testing and mocking did not work for all classes, specifically the classes that used *'Kroy.mainGameScreen…'* which caused a NullPointerException error. An explanation for the error, what we tried and did to circumvent it, and all additional documentation for testing can be found here: https://npstudios.github.io/testing/#assessment-3

DicyCat seemed to take a black box approach, as they have shown that their testing was largely made up from playing the game, rather than analysing the code itself. Differently, we took a more structural testing approach. This meant we could test the game more thoroughly, achieving a strongly tested code base. Although this method of testing is more time consuming, we believe it is a very suitable approach since we will be presenting it for another group to choose, and the more thoroughly tested it is the more attractive it will be. This method of testing is also more reliable and suitable for long term development. We have also implemented some boundary tests where appropriate, which is important as it means we have tested every possible input and avoids any bugs of this kind in the final release.

In terms of the presentation of tests and other testing materials, we are going to take a similar approach as DicyCat. They produced a traceability matrix as well as a test table documenting each test. We have done the same with our JUnit tests, while also providing corresponding documentation within the code, and new playtesting to ensure full coverage of both DicyCat's code base and our additions.

We think a traceability matrix is beneficial as it allows us to clearly see which requirements have been implemented and where. It means that we can be certain that we have fulfilled the customer's requirements and give proof via showing the implementation has been tested. The test table is also a thorough way to document our testing. It allows other developers, for example for the next group who choose this project, to see what has been tested, where, how and why.

DicyCat didn't include any testing statistics such as what their test coverage statistics were. This is clearly something we have changed as using JUnit we are able to gain a line coverage statistic as well as method coverage from testing. While not all-encompassing method to test quality, it does give a strong indication. We analysed the execution duration, which shows the quality of the tests written, and improving on tests that have a long duration. Then we went through how well the Requirements were covered in comparison to tests written, to show how strong the coverage is. All this analysis and statistics can be found here on our website: https://npstudios.github.io/testing/#statistics. You can access all of our testing materials, including our updated traceability matrix, our updated table of tests and our executable tests here https://npstudios.github.io/testing/#assessment-3.

Changes to Methods and Plans

We made no changes to the software development method that DicyCat chose. We decided to continue to use their agile methodology and SCRUM because it is the most appropriate for this type of project. We had already used this methodology ourselves for the past assessments and found no issues with it and we were all comfortable with the process, therefore there was no need to change it. We also found their sprint length of one week to be suitable. It gives each group member plenty of time to get their tasks done whilst not being too long so that issues occur from not reviewing tasks regularly, so we did not change this approach.

In terms of tools, we haven't made any changes either. They used GitHub for version control of code, Google Drive and Google Docs for documentation and Trello for planning tasks. We used these three tools ourselves for past assessments and they worked well therefore we thought there is no need to change it.

The way DicyCat approached team management was very different to how we had approached it in past assessments, so we did make a few changes to this. Firstly, they decided that team members would be able rotate through different roles each week. Due to the small timeframe we have to complete this assessment, we thought it best to stick with the roles we have had previously used which were allocated based on skill and experience. If we were to switch roles it may have delayed progress slightly. For example, one of our team members had much more experience in unit testing so therefore it made more sense for us as a team to allocate that person to testing, reducing the time spent where other members would have had to learn a new concept. However, if a team member had finished their section earlier than expected we would allocate them to another task, allowing us to make sure everything is completed on time or early, giving us more time to quality check documentation and code.

Secondly, DicyCat said that they used Discord as a method of communication, including having meetings over a group call. This, again, is something not suitable to our group as not all of our team members use Discord. Therefore, instead of teaching other members how to use discord, which they are unlikely to check for messages frequently if they are not used to using it, we instead continued to use Facebook Messenger for contact outside of meetings. However, we believe face to face meetings are still the most effective as we can help one another more efficiently should we face any difficulties; Facebook messenger was mainly used to organise meetings and answer quick questions when people were completing independent work.

The first change we made to their plan for assessment four was adding some new tasks which were proofreading, updating the website and submitting the assessment. We thought it was important to include these key closing tasks to ensure we allow the extra time needed for them.

We also changed the start and end date for the assessment as a whole. DicyCat planned to start the assessment on the 18th of February. We believed this is too late as this is when we have our presentation in the weekly practical, therefore we changed this date to be the 14th of February, allowing us plenty of time to create and practice our presentation. They planned to finish the assessment on the 27th of April. We believed this was too close to the deadline, therefore we changed to the 24th so we would have a few extra days of contingency time if any risks occur.

Finally, we changed the planned duration of a lot of tasks. We believed DicyCat had allocated too much time for certain tasks in comparison to others. They allowed 22 days for the Evaluation and Testing Report but only 11 days to implement the changes. As we don't know what changes are yet, we thought it best to allocate more time to this in case it was more complex to implement the changes than previously expected. We also believed that we would be able to get the report writing done in a shorter amount of time as this is what we have achieved in previous assessments. You can access our updated plan for assessment here:
https://npstudios.github.io/files/UpdatedAssessment4Plan.pdf

Appendix

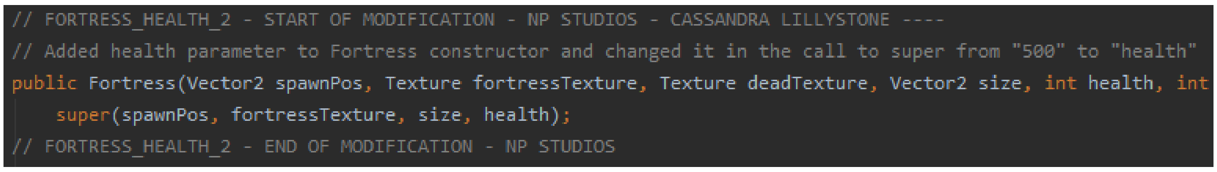
Figure 1: Comment Template

```
// [MODIFICATION_ID] - START OF MODIFICATION – NP STUDIOS – [TEAM_MEMBER_NAME]

// New/Changed fully commented code, with reasons for change

// [MODIFICATION_ID] - END OF MODIFICATION – NP STUDIOS
```

Figure 2: Comment Example



```
// FORTRESS_HEALTH_2 - START OF MODIFICATION - NP STUDIOS - CASSANDRA LILLYSTONE ----
// Added health parameter to Fortress constructor and changed it in the call to super from "500" to "health"
public Fortress(Vector2 spawnPos, Texture fortressTexture, Texture deadTexture, Vector2 size, int health, int
    super(spawnPos, fortressTexture, size, health);
// FORTRESS_HEALTH_2 - END OF MODIFICATION - NP STUDIOS
```